Multi-Output Learning for Predicting Evaluation and Reopening of GitHub Pull Requests on Open-Source Projects

Peerachai Banyongrakkul and Suronapee Phoomvuthisarn
Department of Statistics, Chulalongkorn University, Bangkok, Thailand

Keywords: Pull-Based Development, Pull Request, GitHub, Deep Learning, Multi-Output Learning, Classification.

Abstract: GitHub's pull-based development model is widely used by software development teams to manage software complexity. Contributors create pull requests for merging changes into the main codebase, and integrators review these requests to maintain quality and stability. However, a high volume of pull requests can overwhelm integrators, causing feedback delays. Previous studies have built predictive models using traditional machine learning techniques with tabular data, but these may lose meaningful information. Additionally, relying solely on acceptance and latency predictions may not be sufficient for integrators. Reopened pull requests can add maintenance costs and burden already-busy developers. This paper proposes a novel multi-output deep learning-based approach that early predicts acceptance, latency, and reopening of pull requests, effectively handling various data sources, including tabular and textual data. Our approach also applies SMOTE and VAE techniques to address the highly imbalanced nature of the pull request reopening. We evaluate our approach on 143,886 pull requests from 54 open-source projects across four well-known programming languages. The experimental results show that our approach significantly outperforms the randomized baseline. Moreover, our approach improves accuracy by 8.68%, precision by 1.01%, recall by 11.49%, and F1-score by 6.77% in acceptance prediction, and MMAE by 6.07% in latency prediction, while improving balanced accuracy by 9.43%, AUC by 9.37%, and TPR by 30.07% in reopening prediction over the existing approach.

1 INTRODUCTION

In recent decades, open-source software projects have adopted the pull-based development model (Bird et al., 2009), enabled by GitHub[1] to allow *contributors* to make software changes in a flexible and efficient manner (Gousios et al., 2016) via a *pull request*. The project's *integrators* are responsible for evaluating the pull request and deciding whether to accept or reject the changes. The role of integrators is crucial in the pull-based model (Dabbish et al., 2013) because they must not only make important decisions but also ensure that pull requests are evaluated in a timely matter. In popular projects, the volume of incoming pull requests is too large (Tsay et al., 2014). Therefore, it may increase the burden on already-busy integrators (Gousios et al., 2015) and cause contributors to experience delayed feedback (Gousios et al., 2016).

Thus, several studies using machine learning and statistical techniques have been proposed to support the pull-based model, especially integrators. There have been two main ways to study the pull request evaluation, consisting of the decision to merge (i.e., *acceptance*) and the merging time (i.e., *latency*). Most works have investigated factors influencing acceptance (Gousios et al., 2014; Tsay et al., 2014; Soares et al., 2015; Ortu et al., 2020; Zhang et al., 2022) and latency (Gousios et al., 2014; Yu et al., 2015; Zhang et al., 2021), while a few works have focused on building a prediction model for acceptance (Nikhil Khadke, 2012; Chen et al., 2019; Jiang et al., 2020) and latency (de Lima Júnior et al., 2021).

Acceptance and latency seem to be insufficient for the pull request evaluation. After a pull request is closed by an integrator, in some cases, it may be opened again for further modification and code review (Mohamed et al., 2018). This pull request is called a *reopened pull request*. Even though reopened pull requests rarely happen (Jiang et al., 2019), they may create conflicts with newly submitted pull requests (McKee et al., 2017), add software maintenance costs, and increase the burden for already-busy developers (Mohamed et al., 2018). Two studies (Mohamed et al., 2018; Mohamed et al., 2020) have developed models for predicting reopened pull requests,

[1] https://github.com/

Banyongrakkul, P. and Phoomvuthisarn, S.
Multi-Output Learning for Predicting Evaluation and Reopening of GitHub Pull Requests on Open-Source Projects.
DOI: 10.5220/0012125200003538
In *Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023)*, pages 163-174
ISBN: 978-989-758-665-1; ISSN: 2184-2833

but they make predictions at the first decision, which may be too late. Integrators would benefit from earlier prediction results to identify pull requests more likely to be reopened and come up with timely solutions. However, early prediction is very challenging due to limited available information. If only common tabular features that are available, similar to existing approaches, are used, it may not be sufficient to create accurate predictions.

In this paper, we introduce a novel multi-output deep learning-based approach that predicts acceptance, latency, and reopening of pull requests at the time of submission. Specifically, the predictions can be generated and provided to integrators as feedback immediately after the pull request is created. We make use of deep learning to focus on automating and enhancing performance while overcoming the limited information available at submission time and the highly imbalanced nature of reopened pull requests. In particular, to tackle the limited information, we incorporate both tabular data and textual data from the pull request description and handle the nature of the text by using various pre-trained models. To overcome the highly imbalanced nature, we employ a combination of SMOTE and VAE techniques.

In addition, we address the relationship between pull request outputs, as previous research has shown that reopened pull requests have lower acceptance rates and longer evaluation times than non-reopened ones (Soares et al., 2015; Jiang et al., 2019) by sharing learning between outputs. Regarding the methodologies, we address a gap in pull request prediction research by using a programming language-specific experimental setting that balances specificity and generalization. It avoids the cold-start problem for new projects and overly generalized models, which has been a limitation in prior research that evaluated models at the project or all-in-one level. Also, previous studies have highlighted the significance of programming language in pull request evaluation (Rahman and Roy, 2014; Soares et al., 2015).

We perform extensive experiments on four widely-known programming languages including, Python, R, Java, and Ruby, along with popular open-source software projects that follow the pull-based development model on GitHub. Our approach outperforms the randomized baseline, achieving impressive results on average. We obtain an accuracy of 0.762, a precision of 0.878, a recall of 0.791, and an F1-score of 0.832 in acceptance prediction, while we yield an MMAE of 1.163 in latency prediction. For reopening prediction, we achieve a balanced accuracy of 0.618, an AUC of 0.689, and a TPR of 0.694. The experimental results demonstrate the effectiveness of our approach in validating the main contribution of this paper. Notably, our approach exhibits significant improvements over an existing approach, with enhancements of 8.68% in accuracy, 1.01% in precision, 11.49% in recall, 6.77% in F1-score, 6.07% in MMAE, 9.43% in balanced accuracy, 9.37% in AUC, and 30.07% in TPR. These findings provide strong evidence that our approach effectively improves the predictive performance in the context of pull request evaluation and pull request reopening.

2 BACKGROUND & RELATED WORK

In this section, we provide background information on the pull-based development model, followed by a comprehensive review of the existing literature.

2.1 Pull Request Workflow

Figure 1 shows GitHub's pull-based development workflow that allows contributors to make changes to an open-source project without sharing access to the main repository. Contributors create forks and make changes locally. When a set of changes is ready to be submitted to the main repository, they are required to create an event, called a *pull request*, to request for review and approval by an integrator. The integrator inspects the changes and provides feedback. The contributor can make additional commits to address feedback before approval. The integrators have the final say in whether to accept or reject pull requests, which can have consequences depending on their experience. While pull requests can enhance the efficiency and flexibility of software development, this workflow can increase the workload for integrators, especially in popular projects. (Gousios et al., 2015).

2.2 Pull Request Lifecycle

There are three states of pull requests on GitHub as shown in Figure 2, including:

- **Open:** the pull request has been proposed by the contributor and is during discussions or waiting for the integrator's decision on whether it will be accepted or rejected.
- **Merged:** the integrator approves the changes in the pull request and merges them with the main branch, thus closing the pull request.
- **Closed:** the integrator is not satisfied with the changes and closes the pull request by rejecting it.

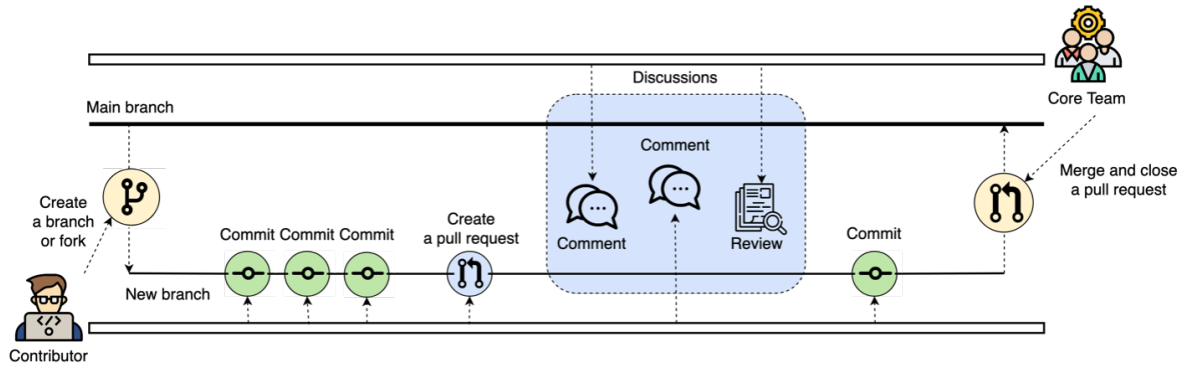

Figure 1: An overview of Github's pull-based development workflow.

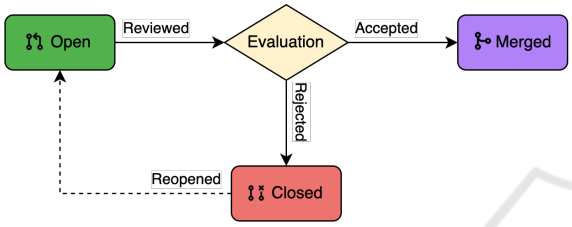

Figure 2: Pull request lifecycle on GitHub.

Pull requests, however, sometimes remain open indefinitely because the integrators are too busy or do not want to discourage the contributor by explicitly rejecting the pull request. In addition to the states above, pull requests can be reopened after being closed when the decision is changed, or further code review is required. The contributor can attempt further updates to reopen the review process, which may lead to a new decision from the integrator. These pull requests are called *reopened pull requests*. Reopening a pull request is considered a risk because it can cause the integrator to take more effort (e.g., add software maintenance costs and increase the burden for an already busy integrator) (Jiang et al., 2019). Moreover, it may cause conflicts with newly submitted pull requests if pull requests are reopened a long time after being closed (McKee et al., 2017). Therefore, the notification of pull request evaluation and pull request reopening can benefit integrators by encouraging timely decisions, prioritizing pull requests, and speeding up the review process, which can lead to accelerated software product development.

2.3 Pull Request Evaluation

Therefore, identifying the quality of pull requests is important, and this is called *pull request evaluation*. Evaluating pull requests is a complex iterative process involving multiple stakeholders. Currently, there are two key aspects of evaluation that researchers study, which are *acceptance* and *latency* (Yu et al., 2015).

2.3.1 Acceptance

Works focusing on the pull request acceptance study the factors influencing the decision of integrators on whether to accept or reject pull requests. For example, the work in (Gousios et al., 2014) found that the acceptance is primarily influenced by whether the pull request modifies recently modified code through Random Forest. With the multidimensional association rule, the work in (Soares et al., 2015) determined factors, including programming languages, that increase the likelihood of a pull request merge. Other works have studied social and technical factors, such as comments affect metrics (Tsay et al., 2014) and contributor experience and politeness (Ortu et al., 2020), using logistic regression model. The work in (Zhang et al., 2022) conducted a comprehensive analysis of factors gathered from a systematic literature review through statistical methods.

Aside from the works focusing on the influencing factors, a few studies concentrate on building a high-quality predictive model. The works in (Nikhil Khadke, 2012) and (Jiang et al., 2020) used machine learning to achieve high accuracy in predicting pull request acceptance, with Random Forest and XGBoost being the most effective algorithms, respectively. The approach by (Jiang et al., 2020) was claimed that it outperformed the previous approach, by (Gousios et al., 2014), which employed Random Forest as their best classifier. Another work is (Chen et al., 2019) which derived new features to build the predictive model from crowdsourcing.

2.3.2 Latency

Researchers focusing on pull request latency explore the factors influencing the latency and estimate the lifetime. The work in (Gousios et al., 2014) divided the pull request lifetime into three classes and used the Random Forest model to study pull request latency, finding that the contributor's merge percentage affects

integration time. Logistic regression was employed in (Yu et al., 2015) to model latency in GitHub projects with continuous integration, and process-related factors were found to be more important when a pull request was closed.

The work in (Zhang et al., 2021) also used linear regression to study latency and found that the relative importance of factors varied depending on the context, with process-related factors being more important when the pull request was closed. The work in (de Lima Júnior et al., 2021) used regression and classification techniques to evaluate pull requests, finding that linear regression worked best for regression while Random Forest was best for classification. Moreover, the relationship between acceptance and latency is studied in (Soares et al., 2015). They found that an increase in the evaluation time for a pull request reduces the chances of its acceptance.

2.4 Pull Request Reopening

To the best of our knowledge, there are only a few works related to reopening pull requests. Mohamed et al. (Mohamed et al., 2018) designed an approach named DTPre to predict reopened pull requests after their first decision, which used oversampling to handle imbalanced datasets. They evaluated DTPre using four different classifiers on seven open-source GitHub projects, finding that Decision Tree with oversampling was the best approach. The recent work (Mohamed et al., 2020) by the same research team, Mohamed et al., extended their work by performing further cross-project experiments on reopened pull request prediction through the same dataset. Their objective was to handle the cold-start problem for new software projects that have a limited number of pull requests. Another study by (Jiang et al., 2019) investigated the impact of reopened pull requests on code review and found that reopened pull requests had lower acceptance rates, longer evaluation time, and more comments than non-reopened ones.

2.5 Gaps in Literature

The existing literature on pull request evaluation has primarily focused on factors influencing acceptance and latency. The studies have examined various technical and social factors using traditional machine learning methods to build predictive models. However, there is limited research on the topic of pull request reopening, which can have a negative impact on software teams. Additionally, there is a lack of models that provide timely predictions for integrators immediately after a pull request is created. Another gap is that text data from the pull request description and the imbalanced nature of reopened pull requests have not been effectively handled. The relationship between pull request outputs, such as acceptance, latency, and reopening, also needs to be addressed. Furthermore, prior research on pull request prediction has typically evaluated models at either the project or all-in-one level, which can result in a cold-start problem for new projects or overly generalized models. Therefore, there is a gap in the literature in terms of using a programming language-specific experimental setting to balance specificity and generalization, which can improve the applicability and relevance of the models in real-world software development scenarios.

3 DATASET

This section describes the dataset used in this empirical study, including the process of data collection and data labeling.

3.1 Overview of Dataset

We used GitHub data from well-known open-source projects developed under popular programming languages. To collect the data and build our dataset, we employed GitHub REST APIs and a web scraping tool. Finally, we filtered the data to derive the final set of data, consisting of 143,886 pull requests (samples) from 54 open-source projects across four programming languages (i.e., Python, R, Java, and Ruby). The pull requests that we collected were created from Aug 2010 to Aug 2022. Table 1 illustrates our dataset, including an overview of programming language characteristics and summarizing the statistical characteristics of the dataset for each language. As can be seen from the table, it appears that we can categorize the languages into two groups: a small community and a big community. Python and R belong to the small community group, while the rest fall into the big one.

3.2 Pull Request Collection

Our data were collected from two sources: the GitHub server and the GitHub website, using GitHub REST APIs and Selenium via Python scripts. We considered only pull requests with a closed status to ensure that they have been decided upon. Initially, we collected the 100 most starred open-source projects written in each programming language. Stargazer counts are commonly used by researchers as a proxy for project popularity (Papamichail et al., 2016). To ensure that our dataset comprised relevant projects, we applied a

Table 1: Descriptive statistical information of our pull request dataset.

Language	Overview		# Pull Requests / Project				
	# Projects	# Pull Requests	Min	Med	Max	Mean	SD
Python	11	9,773	173.00	764.00	2,574.00	888.45	693.12
R	12	8,310	150.00	456.00	1,706.00	755.45	557.19
Java	12	29,202	504.00	1,247.00	6,636.00	2,433.50	2,055.49
Ruby	19	96,601	662.00	3,760.00	13,431.00	5,084.26	3,864.15
TOTAL	**54**	**143,886**					

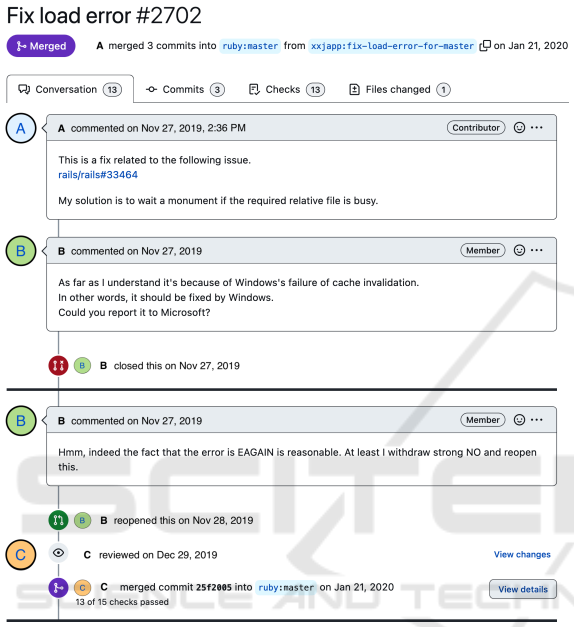

Figure 3: An example of a pull request on GitHub.

second filter based on metrics such as the number of open issues, fork status, number of forks, number of total commits, number of contributors, and number of pull requests. The projects included in our dataset had to meet the following criteria:

- Not a fork version of another project.
- Not a documentation project.
- Have a number of open issues, number of total commits, number of contributors, and number of pull requests greater than or equal to the median of these metrics.

3.3 Pull Request Labeling

Figure 3 displays an example of a pull request in the Ruby project. Due to the purpose of the demonstration, the figure has been edited and we mainly show some important parts of the pull request. The title and body indicate that the contributor, Mr.A, proposed changes to fix an issue related to load error. Initially, the integrator, Mr.B, rejected the pull request as it seemed unrelated to Ruby. However, he later reopened it as some parts seemed reasonable. The pull request was then reviewed by another integrator, Mr.C, who accepted it. This example highlights the risks involved in reopening pull requests when an integrator's initial decision is impaired due to various factors. Thus, early recognition of such risks could help integrators make more effective decisions.

To formulate our predictive problem, we denote t_{pred} as the reference point to the pull request submission time at which a prediction is made for a pull request (i.e., prediction time). We would like to develop a classification approach that can predict three outputs: 1) acceptance, 2) latency, and 3) reopening for a pull request at time t_{pred}. To be more specific, the prediction outcomes would be made using only information available at t_{pred}.

1) **Acceptance:** This reflects the decision of an integrator on whether a pull request is accepted or rejected in the final close. There are two nominal classes for acceptance which are *Accepted*: a pull request is accepted to merge into the main branch and *Rejected*: a pull request is rejected to merge into the main branch.
2) **Latency:** This reflects the time difference between the pull request submission and the final close (i.e., lifetime). We employ the way to discretize the lifetime from (de Lima Júnior et al., 2021) where the magnitude is maintained. We classify latency into five ordinal classes which are *Hour*: lifetime $\leq$ 60 mins, *Day*: 60 mins $<$ lifetime $\leq$ 24 hours, *Week*: 24 hours $<$ lifetime $\leq$ 7 days, *Month*: 7 days $<$ lifetime $\leq$ 4 weeks, and *GTMonth*: lifetime $\geq$ 4 weeks.
3) **Reopening:** this reflects the reopening status of a pull request, showing whether it has been reopened. The reopening task can be considered a problem of anomaly detection due to the highly imbalanced nature of the data. In this context, the number of instances in the positive class (i.e., pull requests that are likely to be reopened) is much

smaller than the number of instances in the negative class (i.e., pull requests that are likely to be closed). There are two nominal classes for the reopening output which are *Reopened*: a pull request has been reopened at least once and *NonReopened*: a pull request has never been reopened.

In the case of the pull request ID 2702, t_{pred} is 2:36 PM, 27 Nov 2019. The information available at 2:36 PM, 27 Nov 2019 is used to predict three outcomes of this pull request which are *Accepted* for the acceptance output, *GTMonth* for the latency output, and *Reopened* for the reopening output.

4 OUR APPROACH

This section presents our proposed approach for predicting pull request evaluation and reopening. We will discuss overview of our approach and two main processes which are feature extraction and modeling.

4.1 Overview of Our Approach

Our proposed approach is a deep learning-based classification approach for predicting acceptance, latency, and reopening of pull requests. Figure 4 shows an overview framework of our approach, which is divided into two phases: the training phase and the execution phase. The training phase involves using historical pull requests to build predictive models. To extract features, we categorize the information of a pull request into two groups: *tabular data* (represented in blue color) and *textual data* (represented in green color). Tabular data is structured data that can be extracted using common feature extraction techniques, resulting in numerical or categorical features. Textual data is unstructured data that require advanced learning techniques to extract meaningful features. Then, oversampling is performed to handle the imbalanced data in the reopening task. Our approach utilizes both types of data (X) along with their corresponding outcomes (Y) to train predictive models using deep learning techniques. The execution phase involves employing the trained models from the training phase to predict three outcomes: acceptance, latency, and reopening, for a new pull request. From the fact that reopening always occurs before pull request evaluation and may have an impact on the other outcomes, our approach predicts the reopening output first. This predicted reopening output is then used as a feature along with the other input features to predict the acceptance and latency of the pull request.

4.2 Feature Extraction

Our approach incorporates two types of feature: *tabular features* and *textual features*. These features play a crucial role in capturing relevant information from pull requests and facilitating accurate prediction.

4.2.1 Tabular Features

A project repository and a pull request contain many valuable attributes that can be extracted and utilized as features to characterize the pull request. Common feature extraction techniques, such as counting, summation, subtraction, and ratio calculation, are deployed based on the attributes of the pull request. The set of tabular features used in our approach is derived from the features employed by previous works related to prediction (Gousios et al., 2014; Jiang et al., 2020; Mohamed et al., 2018; Mohamed et al., 2020; de Lima Júnior et al., 2021). It is worth noting that the features are extracted at the time of pull request submission (t_{pred}), so certain features that appear after submission are not available, such as the number of comments and the number of participants.

4.2.2 Textual Features

A pull request usually contains two pieces of textual information: the title and the body. Contributors use these to summarize and describe the proposed changes. For example, in pull request ID 2702, the title is “Fix load error” and the body is “This is a fix related to the following issue. rails/rails#33464 My solution is to wait a monument if the required relative file is busy.” The title and body can reflect the nature of a pull request, such as the details of a review task and the complexity of the task. Therefore, a well-crafted title and body can reduce the integrator’s effort in executing the review task. However, they have not been taken seriously for use as a pull request predictor in the past. Thus, we use the text as one of our features to characterize our pull request.

In order to use text in machine learning, it must be converted into numerical vectors. Traditional methods like Bag of Words (BoW), N-Gram, and Term Frequency-Inverse Document Frequency (TF-IDF) suffer from sparsity and lose the sequential nature of text (Jurafsky and Martin, 2009). Advanced deep learning techniques, such as pre-trained word embeddings, are capable of handling sequential data with complex dependencies. To address this task, we consider three state-of-the-art pre-trained word embeddings: *Word2Vec* (Mikolov and Others, 2013), *FastText* (Bojanowski et al., 2016), and *BERT* (Devlin et al., 2019) in this study.

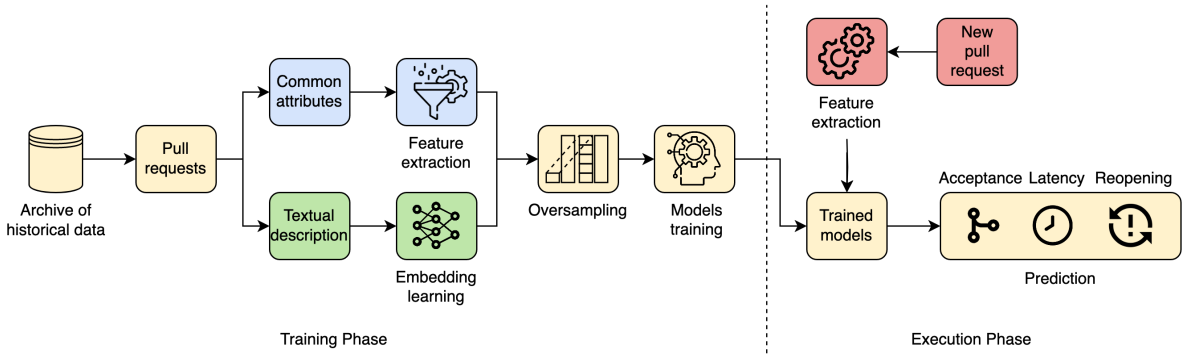

Figure 4: An overview framework of our proposed approach.

Word2Vec, developed by Google, is trained on the Google News corpus and generates meaningful fix-length vector representations for unique words. However, it is context-independent and cannot differentiate the same word in different contexts. FastText, developed by Facebook AI Research, handles subword information and is better for rare or unknown words. It uses N-Grams, character sequences in words, to represent subwords. BERT is a recent transformer-based technique that achieves state-of-the-art performance on several natural language understanding (NLU) tasks. Unlike previous static embeddings, BERT offers context-dependent or semantic embeddings, producing multiple vector representations for a given word based on its surrounding context.

To prepare the input of the pre-trained models, we combine two titles and one body, giving additional weight to the title (i.e., Text = Title + Title + Body). The text inputs undergo preprocessing, such as lowercase conversion, punctuation cleansing, and tokenization. During embedding, a fix-length vector representation is generated for each token. Ultimately, we employ the average pooling technique to derive the final vector representation of the entire text.

4.3 Modeling

To simulate the real situation and address the relationship between pull request reopening and evaluation, we separate modeling into two main stages: the reopening stage and the evaluation stage. More precisely, the reopening output is predicted first, and it is used as one of the features to predict the pull request evaluation.

4.3.1 Reopening Stage

In the reopening stage, we follow a three-stage process of feature extraction, oversampling, and classification (see Figure 5). We begin by combining tabular and textual features extracted by pre-trained word embedding, as detailed in the previous section. However, since the data is highly imbalanced, with a majority of non-reopening samples and a minority of reopening samples, we use the *Variational Autoencoder (VAE)* to generate additional reopening samples in order to achieve balance with the non-reopening samples.

VAE is a generative model that can learn to approximate a probability distribution of input data by encoding them into a lower-dimensional latent space and then decoding them back to the original data space. By sampling from the learned latent space, VAE can generate new data points that are similar to the original data, effectively increasing the size of the dataset. To ensure that we have enough reopening samples for training the VAE, we first use the *Synthetic Minority Over-sampling Technique (SMOTE)* to upsample the positive class (i.e., *Reopened* class). Next, we train our VAE exclusively on pull requests with the *Reopened* class. We use the decoder part of the trained VAE to generate reopening samples by introducing random noise from a normal distribution. These generated samples are mixed with the original ones and used to train a *deep neural network (DNN)* to predict the probability of reopening as output. The number of samples is balanced through the oversampling in the training set only.

4.3.2 Evaluation Stage

During the evaluation stage, we extract features from the pull request description and other relevant data, similar to the feature extraction process used in the reopening stage (without oversampling). We combine these features with the reopening probability obtained from the reopening stage. A DNN is then trained to predict two outputs: acceptance and latency of the pull request. The approach used in the evaluation stage is depicted in Figure 6.

The architecture utilized for the DNNs in both stages is a very common feedforward neural network,

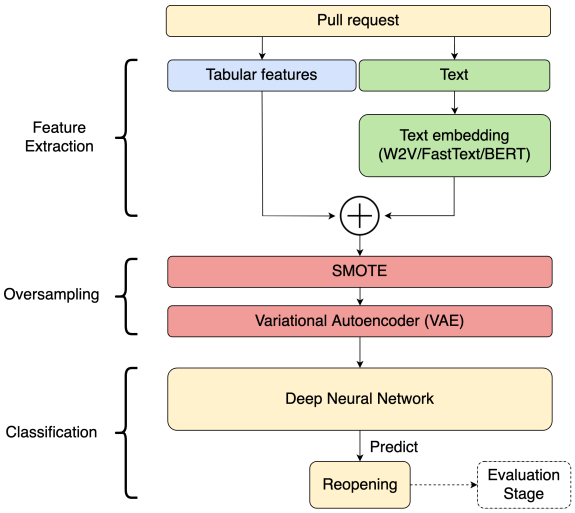

Figure 5: A model architecture of the reopening stage of our approach.

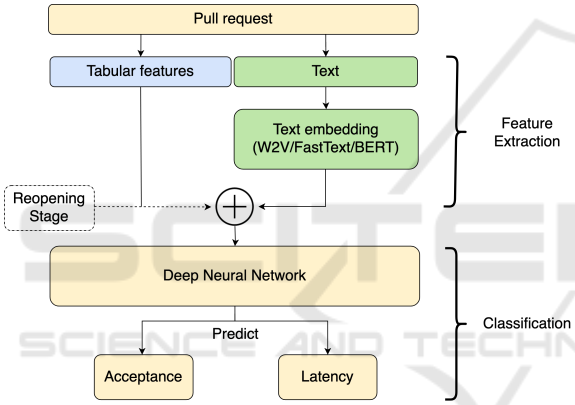

Figure 6: A model architecture of the evaluation stage of our approach.

consisting of an input layer, a normalization layer, followed by multiple blocks of dense layers and dropout layers, and an output layer. The key distinction is that the DNN in the reopening stage solely predicts the reopening output, while the DNN in the evaluation stage is a *multi-output* DNN that predicts the acceptance and latency outputs using shared learning.

5 EVALUATION

In this section, we conduct a comprehensive evaluation of our approach. We will outline the research questions guiding our evaluation, define the performance measures used, and describe the experimental settings employed for the evaluation.

5.1 Research Questions

In this study, we aim to answer two research questions through our empirical evaluation, which seeks to provide insights into the effectiveness of our proposed approach in predicting pull request evaluation and reopening.

RQ1: *Sanity Check (Is the proposed approach suitable to early predict pull request evaluation and pull request reopening?)*

This aims to perform a sanity check on the suitability of our approach in predicting pull request evaluation and reopening at the submission time. To address this, we compare the performance of our approach against a *randomized algorithm*. Conducting a sanity check using a rule-based model is a common practice in software engineering research (Al-Zubaidi et al., 2018; Shepperd and MacDonell, 2012; Sarro et al., 2016). We repeat the random guessing process 5000 times and take the average performance to ensure statistical significance. Our approach should surpass the baseline, which relies on random guessing, to demonstrate its suitability for the early prediction of pull request evaluation and pull request reopening.

RQ2: *Does the proposed approach outperform the existing approach?*

The objective of this research question is to compare the predictive performance between the existing approach and our approach. Due to the fact that no existing approach predicts in the same manner as our proposed approach, we utilize an alternative approach. The alternative approach incorporates tabular features, feature selection techniques, and traditional machine learning-based single-output classifiers, such as *Decision Tree*, *Random Forest*, and *XGBoost*, which have been recognized as the best performers in previous studies (Gousios et al., 2014; Jiang et al., 2020; Mohamed et al., 2018; Mohamed et al., 2020; de Lima Júnior et al., 2021), to represent the existing approaches. The performance of our approach should be better than the existing approach to indicate that textual features extracted from the pre-trained model, our oversampling technique (i.e., SMOTE combined with VAE), shared learning, and deep learning classifiers can overcome the challenges, posed by the limited information available at the time of submission and highly imbalanced data, as well as improve the performance for the prediction of pull request evaluation and pull request reopening.

5.2 Performance Measures

To measure the predictive performance of the approaches, we use common binary classification met-

rics, such as, accuracy, precision, recall, F1-score, and AUC for the acceptance task. However, we applied Macro-Averaged Absolute Error (MMAE) for the latency task because it can assess the distance between an actual class and a predicted one. It can also be applied to the imbalanced multi-class classification because of the macro-averaged technique. MMAE has been used in many research works (Baccianella et al., 2009; Choetkiertikul et al., 2018; Wattanakriengkrai et al., 2019) to tackle the ordinal multi-class classification problem. Note that for the MMAE metric, lower values indicate better performance. The formula of MMAE is defined in Equation 1 where K is a set of classes, $|K|$ is the number of classes, k is a class within K, y^i is the true class, and n_k is the number of true classes with class k. σ is the indicator function.

$$MMAE = \frac{1}{|K|}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{i=1}^{n}|\hat{y}^i - k|\sigma[y^i = k] \quad (1)$$

In addition, we used metrics that can handle highly imbalanced data and enable accurate anomaly detection, such as, balanced accuracy (BA), AUC, True Positive Rate (TPR), and False Negative Rate (FPR) (Kale et al., 2022; Trauer et al., 2021) for the reopening task.

5.3 Experimental Setting

We split the data into three sets: training, validation, and testing, using a *hold-out technique*. Pull requests were sorted by close date to ensure that the model learned only from past data available at the training time. Specifically, the pull requests in the training set and the validation set were closed before the pull requests in the testing set, and the pull requests in the training set were also terminated before the pull requests in the validation set. Since our experiment was specific to programming languages, we developed a separate approach for each language. The small community programming languages used a 60/20/20 split while the big ones used an 80/10/10 split. In our study, the training dataset was used to train our model and we applied the validation dataset to choose the best models as well as to tune hyperparameters. Lastly, the testing dataset was applied to evaluate the performance of our model.

To ensure a fair comparison of performance, all approaches were trained and validated on the same experimental environment using the identical dataset with the shared data splitting. The set of tabular features were also shared between both the existing approach and our approach. They also shared the same tuning hyperparameter technique, which involves using a randomized algorithm with 30 iterations. Furthermore, they used the same performance metrics for model tuning. Specifically, AUC was used for the acceptance and reopening tasks, and MMAE was used for the latency task.

6 RESULTS

This section reports the evaluation results to answer the research questions[2]. Table 2 shows the evaluation results for pull request evaluation and reopening achieved by randomized baseline, existing approach, and our approach in four programming languages.

Results for RQ1: For the pull request evaluation, the analysis of all associated measures (i.e., accuracy, precision, recall, F1-score, AUC, and MMAE) suggests that the predictive results obtained with our approach (Our), are better than those achieved by using the randomized baseline (Randomized) in all cases (24/24) consistently. Our approach improves between 43.74% (in Python) to 65.21% (in Java) in terms of accuracy, 4.71% (in Ruby) to 60.44% (in Java) in terms of precision, 48.31% (in Python) to 65.17% (in Java) in terms of recall, 31.57% (in R) to 62.83% (in Java) in terms of F1-score, 29.75% (in R) to 81.85% (in Java) in terms of AUC, and 24.79% (in R) to 28.84% (in Java) in terms of MMAE over the baseline.

For the pull request reopening task, our approach outperforms the randomized baseline in most cases (14/16) in terms of balanced accuracy, AUC, TPR, and FPR. Our approach improves over the baseline between 17.92% (in Java) to 28.21% (in Ruby) in terms of balanced accuracy, 24.71% (in Java) to 46.39% (in Python) in terms of AUC, and 27.15% (in Ruby) to 52.75% (in R) in terms of TPR, while the results for FPR were mixed, with some cases showing improvement and others showing a decline compared to the baseline. Specifically, our approach improves FPR by 19.96% (in Python) and 29.27% (in Ruby) over the baseline, while it was unable to improve in R and Java. Our approach achieves the best performance in Ruby, as it consistently outperforms the baseline in all evaluation measures.

Our proposed approach outperforms the randomized baseline in all four programming languages, thus our approach is suitable for predicting pull request evaluation and reopening at the submission time.

[2]All the experiments were run on Macbook Pro with macOS Monterey Version 12.4, Apple M1 Pro chip, and 16GB RAM.

Table 2: Evaluation results: Performance comparison between the randomized baseline (Randomized), the existing approach (Existing), and our approach (Our) for predicting the pull request evaluation and reopening, reported as accuracy (Acc), precision (P), recall (R), F1-score (F1), AUC, Macro-Averaged Absolute Error (MMAE), balanced accuracy (BA), True Positive Rate (TPR), and False Positive Rate (FPR).

Language	Approach	Acceptance					Latency	Reopening			
		Acc	P	R	F1	AUC	MMAE	BA	AUC	TPR	FPR
Python	Randomized	0.500	0.771	0.500	0.607	0.500	1.599	0.500	0.500	0.500	0.500
	Existing	0.681	**0.878**	0.681	0.767	0.708	1.270	0.618	0.705	0.500	**0.264**
	Our	**0.719**	0.874	**0.742**	**0.803**	**0.716**	**1.171**	**0.639**	**0.732**	**0.679**	0.400
R	Randomized	0.500	0.835	0.500	0.625	0.500	1.600	0.500	0.500	0.501	**0.500**
	Existing	0.594	0.865	0.609	0.714	0.589	1.224	0.527	0.649	**0.765**	0.709
	Our	**0.721**	**0.874**	**0.777**	**0.823**	**0.649**	**1.204**	**0.600**	**0.715**	**0.765**	0.564
Java	Randomized	0.500	0.523	0.500	0.511	0.500	1.600	0.500	0.500	0.501	0.500
	Existing	0.792	0.815	0.779	0.797	0.872	1.198	0.554	0.584	0.362	**0.254**
	Our	**0.826**	**0.839**	**0.826**	**0.832**	**0.909**	**1.139**	**0.590**	**0.624**	**0.700**	0.515
Ruby	Randomized	0.500	0.883	0.500	0.638	0.500	1.600	0.500	0.500	0.500	0.500
	Existing	0.737	0.920	0.769	0.838	0.682	1.262	0.557	0.583	0.506	0.391
	Our	**0.781**	**0.925**	**0.819**	**0.868**	**0.720**	**1.141**	**0.641**	**0.684**	**0.635**	**0.354**

Results for RQ2: We compare the performance achieved from our approach (Our) against the existing approach (Existing). For the pull request evaluation task, the analysis of all corresponding measures suggests that our approach achieves better performance in most cases (23/24) compared to the existing approach. Our approach improves between 4.33% (in Java) to 21.36% (in R) in terms of accuracy, 0.41% (declining in Python) to 3.01% (in Java) in terms of precision, 5.97% (in Java) to 27.73% (in R) in terms of recall, 3.68% (in Java) to 15.20% (in R) in terms of F1-score, 1.13% (in Python) to 10.20% (in R) in terms of AUC, and 1.69% (in R) to 9.62% (in Ruby) in terms of MMAE.

For the pull request reopening task, our approach outperforms the existing approach in most cases (13/16) in terms of balanced accuracy, AUC, TPR, and FPR. Our approach improves over the baseline between 3.40% (in Python) to 15.00% (in Ruby) in terms of balanced accuracy, 3.89% (in Python) to 17.45% (in Ruby) in terms of AUC, and 0.00% (in R) to 92.00% (in Java) in terms of TPR, while the results for FPR were mixed, with some cases showing improvement and others showing a decline compared to the existing approach. Explicitly, our approach improves FPR by 9.63% (in Ruby) and 20.41% (in R) over the existing approach, while showing a decline in Python and Java. Overall, our approach shows better performance than the existing approach. Ruby is also the programming language where our approach achieves the highest performance, as it consistently outperforms the existing approach in all evaluation measures.

Our proposed approach outperforms the existing approach in all four programming languages. We can, thus, conclude that textual features extracted from the pre-trained models, our oversampling technique, shared learning, and deep learning classifiers improve the performance for prediction of pull request evaluation and pull request reopening.

It is noteworthy that in our reopening experiments, we observed the FPR improvement in 14 cases over the baseline, while we were unable to improve in two cases. Moreover, we observed the FPR improvement in 13 cases over the existing approach, while we were unable to improve in three cases. However, it is crucial to consider that the importance of TPR or FPR may vary depending on the specific application and cost associated with each project. In our study, we have used AUC as the main evaluation metric, which provides a balanced measure between TPR and FPR. This allows us to account for the trade-off between sensitivity and specificity, and strike a balance in our analysis. Based on AUC, our results excel in all cases.

7 THREATS TO VALIDITY

In this section, we will discuss potential threats to the validity of our research findings.

External Validity: Our study provided a broad range of perspectives by analyzing 54 real-world well-known open-source projects across four popular programming languages on GitHub. However, our findings may not be representative of all programming languages and all kinds of software projects, espe-

cially in commercial settings. To address this limitation, we plan to expand our experiment to a more diverse range of projects and languages in the future.

Internal Validity: We minimized bias and errors in our dataset and experiments by considering actual pull request outputs from real integrators. We also processed only the information available at the time of pull request submission (t_{pred}) by scraping the GitHub website, avoiding any potential information leakage.

Construct Validity: We adopted standard evaluation metrics commonly used in classification tasks. The metrics have also been employed in prior software engineering research to assess the effectiveness of different approaches, enabling us to compare and validate our results. However, evaluating the reopening prediction presents a challenge due to highly imbalanced data, and there is limited prior work that addresses this issue. Therefore, we employed common metrics that have been used in other domains to assess our approach's performance on this task.

Conclusion Validity: We took a meticulous and cautious approach when drawing conclusions based on the extracted features from the studied project repositories. However, it should be noted that the latency may not always reflect the actual review and integration time of a pull request, as there may be other factors beyond the integration process such as the integrator having a heavy workload or lack of interaction with the contributor (de Lima Júnior et al., 2021). Additionally, the pull request reopening may not always indicate the actual reopening because it can occur due to accidental closure (Jiang et al., 2019).

8 CONCLUSIONS

In this paper, we have proposed a novel deep learning-based approach to predict pull request acceptance, latency, and reopening in open-source software projects hosted on GitHub. Our prediction is delivered at the time of pull request submission to enable integrators to plan their work more effectively, especially in large projects. Our approach combines both tabular and textual features to capture relevant information. We leverage the state-of-the-art pre-trained models to extract the meaningful vector representation of textual data while we utilize the SMOTE combined with VAE as the oversampling technique. In addition, our approach incorporates shared learning and deep neural networks to address the gaps and the challenges and to improve the predictive performance for the prediction of pull request evaluation and pull request reopening.

We have conducted an extensive evaluation on four well-known programming languages, which demonstrated that our approach significantly outperforms random guessing, and shows the advantages of our approach over the existing approach. In terms of future work, we plan to validate our approach with a wider range of programming languages along with larger projects, especially those in industrial settings. We aim to explore new sources of information that can better characterize pull requests, such as code changes, to enhance the predictive performance, particularly for the reopening task. We plan to take the next step in the development of our approach by integrating it as a tool within the GitHub platform. This will allow us to gather feedback from real users and enable future analysis and refinement of the approach.

REFERENCES

Al-Zubaidi, W. H. A., Dam, H. K., Choetkiertikul, M., and Ghose, A. (2018). Multi-Objective Iteration Planning in Agile Development. *Proceedings of Asia-Pacific Software Engineering Conference (APSEC)*, 2018-Decem:484–493.

Baccianella, S., Esuli, A., and Sebastiani, F. (2009). Evaluation Measures for Ordinal Regression. In *Proceedings of 9th International Conference on Intelligent Systems Design and Applications*, pages 283–287, Pisa, Italy.

Bird, C., Rigby, P. C., Barr, E. T., Hamilton, D. J., German, D. M., and Devanbu, P. (2009). The promises and perils of mining git. In *Proceedings of 6th IEEE International Working Conference on Mining Software Repositories*, pages 1–10, Vancouver, BC, Canada.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Chen, D., Stolee, K., and Menzies, T. (2019). Replication can improve prior results: A github study of pull request acceptance. In *Proceedings of IEEE International Conference on Program Comprehension*, volume 2019-May, pages 179–190, Montreal, QC, Canada. IEEE Computer Society.

Choetkiertikul, M., Dam, H. K., Tran, T., Ghose, A., and Grundy, J. (2018). Predicting Delivery Capability in Iterative Software Development. *IEEE Transactions on Software Engineering*, 44(6):551–573.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. (2013). Leveraging transparency. *IEEE Software*, 30(1):37–43.

de Lima Júnior, M. L., Soares, D., Plastino, A., and Murta, L. (2021). Predicting the lifetime of pull requests in open-source projects. *Journal of Software: Evolution and Process*, 33(6).

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis,*

MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics.

Gousios, G., Pinzger, M., and Deursen, A. V. (2014). An exploratory study of the pull-based software development model. In *Proceedings of International Conference on Software Engineering*, number 1 in ICSE 2014, pages 345–355, Hyderabad, India. IEEE Computer Society.

Gousios, G., Storey, M. A., and Bacchelli, A. (2016). Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of International Conference on Software Engineering*, volume 14-22-May-2016, pages 285–296, Austin, TX, USA. IEEE Computer Society.

Gousios, G., Zaidman, A., Storey, M.-A., and van Deursen, A. (2015). Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *Proceedings of 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 358–368, Florence, Italy.

Jiang, J., Mohamed, A., and Zhang, L. (2019). What are the Characteristics of Reopened Pull Requests? A Case Study on Open Source Projects in GitHub. *IEEE Access*, 7:102751–102761.

Jiang, J., teng Zheng, J., Yang, Y., and Zhang, L. (2020). CTCPPre: A prediction method for accepted pull requests in GitHub. *Journal of Central South University*, 27(2):449–468.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA.

Kale, R., Lu, Z., Fok, K. W., and Thing, V. L. L. (2022). A Hybrid Deep Learning Anomaly Detection Framework for Intrusion Detection. In *Proceedings of IEEE 8th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pages 137–142, Jinan, China.

McKee, S., Nelson, N., Sarma, A., and Dig, D. (2017). Software Practitioner Perspectives on Merge Conflicts and Resolutions. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 467–478.

Mikolov, T. and Others (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, pages 1–9.

Mohamed, A., Zhang, L., and Jiang, J. (2020). Cross-project reopened pull request prediction in github. In García-Castro, R., editor, *Proceedings of The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*, pages 435–438, USA. KSI Research Inc.

Mohamed, A., Zhang, L., Jiang, J., and Ktob, A. (2018). Predicting Which Pull Requests Will Get Reopened in GitHub. In *Proceedings of Asia-Pacific Software Engineering Conference (APSEC)*, volume 2018-Decem, pages 375–385. IEEE Computer Society.

Nikhil Khadke, Ming Han Teh, M. S. (2012). Predicting Acceptance of GitHub Pull Requests.

Ortu, M., Destefanis, G., Graziotin, D., Marchesi, M., and Tonelli, R. (2020). How do you Propose Your Code Changes? Empirical Analysis of Affect Metrics of Pull Requests on GitHub. *IEEE Access*, 8:110897–110907.

Papamichail, M., Diamantopoulos, T., and Symeonidis, A. (2016). User-Perceived Source Code Quality Estimation Based on Static Analysis Metrics. In *Proceedings of 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 100–107.

Rahman, M. M. and Roy, C. K. (2014). An insight into the pull requests of GitHub. In *Proceedings of 11th Working Conference on Mining Software Repositories (MSR 2014)*, pages 364–367. Association for Computing Machinery.

Sarro, F., Petrozziello, A., and Harman, M. (2016). Multi-objective software effort estimation. In *Proceedings of International Conference on Software Engineering*, volume 14-22-May-, pages 619–630.

Shepperd, M. and MacDonell, S. (2012). Evaluating prediction systems in software project estimation. *Information and Software Technology*, 54(8):820–827.

Soares, D., Limeira, M., Murta, L., and Plastino, A. (2015). Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1541–1546, New York, NY, USA. Association for Computing Machinery.

Trauer, J., Pfingstl, S., Finsterer, M., and Zimmermann, M. (2021). Improving production efficiency with a digital twin based on anomaly detection. *Sustainability (Switzerland)*, 13(18).

Tsay, J., Dabbish, L., and Herbsleb, J. (2014). Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of International Conference on Software Engineering*, number 1 in ICSE 2014, pages 356–366, Hyderabad, India. IEEE Computer Society.

Wattanakriengkrai, S., Srisermphoak, N., Sintoplertchaikul, S., Choetkiertikul, M., Ragkhitwetsagul, C., Sunetnanta, T., Hata, H., and Matsumoto, K. (2019). Automatic Classifying Self-Admitted Technical Debt Using N-Gram IDF. In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*, volume 2019-Decem, pages 316–322.

Yu, Y., Wang, H., Filkov, V., Devanbu, P., and Vasilescu, B. (2015). Wait For It: Determinants of Pull Request Evaluation Latency on GitHub. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 367–371.

Zhang, X., Yu, Y., Gousios, G., and Rastogi, A. (2022). Pull Request Decision Explained: An Empirical Overview. *IEEE Transactions on Software Engineering*, 49.

Zhang, X., Yu, Y., Wang, T., Rastogi, A., and Wang, H. (2021). Pull Request Latency Explained: An Empirical Overview. *Empirical Software Engineering*, 27.